Statistical Techniques in Robotics (16-831, S21) Lecture #22 (Monday, April 26)

Imitation Learning & LP-IRL

Lecturer: Kris Kitani *Scribes: Zihang Lai*

1 Review

In the last lectures, on the topic of Reinforcement Learning (RL), we had analyzed three main types of Reinforcement Learning algorithms: (1) Policy-based, (2) Value-based, and (3) Hybrid. We discussed some key algorithms for each type of algorithms, including Policy Iteration, Value Iteration, TD Learning, Policy Gradient, Actor-Critic methods and so on. In this lecture, we will move to a different topic of Imitation Learning. We will discuss the high level idea of Imitation Learning and a specific type of IL algorithm called Inverse Reinforcement Learning (IRL). For a quick high level takeaway, we note that RL requires a predefined reward function, which could be hard to set by hand. For instance, it might be complex to define a reward function for a robotic arm that aims to grasp things. Should the controlling policy gets penalised if a specific torque is applied? Or when the arm rotates away from the object that it aims to grasp? It could be easier to just show examples. We also note that because RL algorithms learn from interacting with the environment, they could take a long time to learn if the state space is too large.

This lecture, after the review section, will discuss an approach to address these issues: Imitation Learning, and specifically, the Inverse Reinforcement Learning algorithm. In the next two subsections, we first recap the Reinforcement Learning algorithms and then motivates Imitation Learning.

1.1 Reinforcement Learning

1.1.1 Problem set-up

The Reinforcement Learning algorithms are used to solve the optimal policy for a Markov Decision Process. The Markov Decision Process (MDP) is defined by a tuple (S, A, P, r, H, ρ), where S is the state space, A is the action space, P is the transition dynamics where $P(s'|s, a)$ is the probability of reaching state s' from state s when action a is taken; r is the reward function, H is the planning horizon, and ρ is the probability distribution of the first state.

A policy $\pi(a|s)$ defines the probability of taking action a at state s. A trajectory is defined by the execution of a policy from an initial state to the plaanning horizon: $\tau = \{s_1, a_1, s_2, ..., s_H, a_H, s_{H+1}\}$ where $s_1 \sim \rho$, $a_t \sim \pi(\cdot|s_t)$ and $s_{t+1} \sim \pi(\cdot|s_t, a_t)$. Given a MDP, an RL algorithm aims to find the optimal policy that yields the best total rewards $R(\tau) = \sum_t r(s_t, a_t)$, or some discounted form, from the initial state.

1.1.2 Value-based vs. Policy-based

We recall the three different types of Reinforcement Learning algorithms:

1. Value-based: By learning from experience, we estimate a value function that scores a particular state or state-action pair and then use this value function to improve the policy.
2. Policy-based: Use the reward from interacting with the environment to directly optimize for the policy, without relying on a value function.
3. Hybrid: A hybrid of value-based methods and policy-based methods.

Clearly, although the distinctions between the above three types of algorithm is clear, they share a common feature that the RL algorithms has to learn everything from experience. Imitation Learning algorithms, which we discussed in this lecture, allows learning from behavior demonstrated by an expert policy. Therefore, it could enable faster and better learning of the optimal policy.

1.1.3 Model-based vs. Model-free

We also recall two different assumptions that Reinforcement Learning algorithms make:

1. Model-based: The algorithm assumes knowing the transition dynamics function, $P(s'|s,a)$, and the reward function, $r(s,a,s')$.
2. Model-free: The algorithm does not make use of the transition function and the reward function.

In Imitation Learning, there are also model-based algorithms, which requires dynamics function and reward function, and model-free algorithms, which does not depend on the model. In this lecture, we discussed the Inverse Reinforcement Learning algorithm. In IRL, we solve for the reward function, therefore the reward function of the model is not known to the IL algorithm. However, the two scenarios of the IRL we discussed in this lecture assumes known transition dynamics.

2 Summary

2.1 Imitation Learning Definitions

As the name suggests, the high level idea of Imitation Learning is learning from expert demonstrations or policies, which makes it different from Reinforcement Learning, which only learns from experience. To formulate the Imitation Learning framework, we introduce the formal definition of IL:

$$A_{IL} : \langle D^*, \pi^*, T, R \rangle \rightarrow \pi$$

- **Expert demonstrations**

$$D^* = \{\zeta_n\}_{n=1}^N \sim \varepsilon|\pi^* \tag{1}$$

  Expert demonstrations are trajectories sampled from the optimal policy. The IL algorithms leverages these expert demonstrations to learn a good policy. In another word, The best reward function could also be interpreted as the policy that best *explains* the given demonstrations. The expert demonstrations could be sequence of state: $\zeta = \{s_0, s_1, ...s_T\}$, sequence

of state-action pairs: $\zeta = \{s_0, a_0, s_1, a_1, ..., a_{T-1}, s_T\}$, or some other forms, depending on the specific algorithm. One real world example discussed in the lecture is learning optimal policy for a high dimensional manipulator arm. The expert demonstration could be the trajectory obtained from a grad student physically moving the robot arm so that it moves to a certain configuration.

- **Optimal policy**

$$\pi^*(a|s) \tag{2}$$

Optimal policy is the oracle policy that IL algorithm tries to learn and the expert demonstration samples from. Note that the optimal policy may or may not be available, depending on the algorithm. Specifically, only in a class of IL algorithm, namely, Interactive IL, is the optimal policy available. In Interactive IL, the optimal policy can be queried from whenever the algorithm needs. So the optimal policy can be thought of as a *coach* that gives instructions in a specific state. Note also that even when the optimal policy is available, it can only be *sampled from*, but not known in its parametric form. The inner work of the policy is still not provided to the IL algorithm. This makes sense because if the parameters of the optimal policy is given, the algorithm only needs to copy from the given policy and does not to learn anything.

- **Dynamics function**

$$P(s'|s, a) \tag{3}$$

Dynamics function is defined the same way as in Reinforcement Learning: the probability of reaching state s' from state s with action a taken. Similar to the expert demonstration D and the oracle policy π^*, the dynamics function is not always given. For example, passive IL algorithms, *e.g.* the behaviour cloning algorithm, only makes use of demonstration and does not assume a known dynamics function.

- **Reward function**

$$R(s, a, s') \tag{4}$$

Reward function is also defined the same way as in Reinforcement Learning: in the most general form, the reward function defines what the agent achieves at a particular state s, taking action a and reaches next state s'. In the two algorithms (IRL algorithm 1 and 2) we covered in this lecture, we aim to find the best reward function and does not have a given reward function.

Moving ahead, we define three types of Imitation Learning algorithms.

2.2 Types of Imitation Learning Algorithms

Table 1 shows three types of Imitation Learning algorithms, based on different input information. In this section, we give a rough definition for each category of algorithms.

2.2.1 Passive IL

Passive Imitation Learning (commonly called the *behavior cloning* algorithm) only makes use of demonstrations to learn the policy. The set up is similar to a supervised learning algorithm: we

	Passive IL	Active IL	Interactive IL
Demostration D^*	yes	yes	optional
Environment ε	no	yes	yes
Oracle π^*	no	no	yes
Dynamics T	no	optional	optional
Reward R	no	optional	optional

Table 1: Three types of Imitation Learning algorithms.

optimized for the policy so that the behavior of the policy is close to the demonstration:

$$\arg\min_{\theta} \sum_{\tau=\{s_t,a_t\}_{t=1}^{T}} l(\pi_\theta(a|s_t), a_t) \tag{5}$$

As the name suggests, this approach clones the behavior of the expert demonstration by treating it as input-label pairs. Passive IL does not interact with the environment in any form.

2.2.2 Active IL

Active IL also gets expert demonstrations but also have access to the environment. Optionally, active IL algorithms have access to dynamics function and/or reward function. This is similar to the model-based and model-free distinction we have encountered in RL algorithms. In the next section, we will discuss Inverse Reinforcement Learning, which is a form of Active IL. IRL could have access to the dynamics function, but it does not make use of the reward function.

2.2.3 Interactive IL

Interactive IL does not always have access for demonstration, but instead, it could have sample feedback from the oracle policy π^* when the algorithm is interacting with the environment. The algorithm act in the environment, and when needed, ask for the oracle feedback to estimate the optimal policy. Interative IL provides *sequential instructive* feedback, which is different from other learning algorithms we discussed earlier in this course (For example, RL provides *sequential* but *evaluative* feedback; PWEA provides *instructive* but *one-shot* feedback).

2.3 Inverse Reinforcement Learning (IRL)

Inverse Reinforcement Learning is a type of Imitation Learning algorithm. It is an Active IL algorithm which leverages both the expert demonstration and environment to learn, but it does not have the reward function. The objective of IRL can be interpreted as finding the optimal reward function that best *explains* the expert demonstration. Once this reward function is found, it can be used to estimate a policy π with a standard RL algorithm.

2.3.1 Why IRL?

Model-based Reinforcement Learning algorithms estimate the optimal policy from a known dynamics model and a known reward function, but the reward function could be difficult to define. Furthermore, understanding why we take an action (reward function) could be more beneficial to learning a policy under a specific environment. This is because the learned knowledge (reward function) could generalize and be transferred into unseen environment.

2.3.2 Three LP-IRL Algorithms

Depending on the assumptions made, LP-IRL [1] has three scenarios:

1. IRL for finite state spaces
2. IRL for large state spaces
3. IRL from sampled trajectories

Each scenario involves solving a linear programming problem, therefore the name of LP-IRL. In this lecture, we will only cover the first two scenarios (algorithm 1 and 2).

2.3.3 IRL for finite state spaces (algorithm 1)

Goal: Find the reward function(s) such that the policy is optimal.

Assumptions:

- State space is finite.
- Transition model is known.
- Complete policy is known.

IRL for finite state spaces is the most easy case in the three LP-IRL algorithms with most assumptions. Moreover, these assumptions are usually not satisfied. For example, in real problems, the state space is often infinite, and the transition model is not known, and the complete policy is certainly not accessible. However, getting started from IRL for finite state spaces, we can better understand the ideas of IRL and the assumptions will be removed later.

In order to set up the objective function for optimizing reward function R, we consider the reward function that leads to $Q^\pi(a, s)$. Intuitively, we want the optimal policy to achieve highest accumulative reward, and the difference between the cumulative reward of the best action and the cumulative reward of the second best action should be as large as possible.

Following this intuition, we can define the objective function as follows:

$$\max_R \{\sum_s Q^\pi(s, a^*) - \max_{a \neq a^*} Q^\pi(s, a)\} \tag{6}$$

where $a*$ is the optimal action and $Q^\pi(s,a)$ is the state-action value function defined as

$$Q^\pi(s,a) = R(s) + \gamma \sum_{s'} p(s'|s,a)V^\pi(s') \tag{7}$$

$$V^\pi(s) = \sum_a \pi(a|s)Q(s,a) \tag{8}$$

However, naively optimize for Eq. 6 could lead to solutions where $Q^\pi(s,a^*)$ goes to infinity and other Q values goes to negative infinity because R is unbounded. Therefore, we add an extra regularization term to the objective function to penalize large reward values. The final objective function is the following:

$$\max_R \{\sum_s Q^\pi(s,a^*) - \max_{a \neq a^*} Q^\pi(s,a)\} - \lambda||R||_1 \tag{9}$$

Linearity in matrix form: First we express the Q functions

$$Q^\pi(s,a) = R(s) + \gamma \sum_{s'} p(s'|s,a)V^\pi(s') \ \forall s,a \tag{10}$$

in matrix form:

$$\mathbf{Q}^\pi(a) = (\mathbf{R} + \gamma \mathbf{P}_a \mathbf{V}^\pi) \ \forall a \tag{11}$$

where we reduce the number of equations from $s \times a$ to a by expressing all the equations for a specific action in a matrix.

Similarly, we could write the linear form of the V function

$$V^\pi(s) = R(s) + \gamma \sum_{s'} p(s'|s,\pi(s))V^\pi(s') \ \forall s \tag{12}$$

as follows:

$$\mathbf{V}^\pi = (\mathbf{R} + \gamma \mathbf{P}_a \mathbf{V}^\pi) \tag{13}$$

$$\mathbf{V}^\pi = (\mathbf{I} - \gamma \mathbf{P}_a)^{-1}\mathbf{R} \tag{14}$$

Next, we write the difference in Q-values in linear form:

$$\mathbf{Q}^\pi(a^*) - \mathbf{Q}^\pi(a) = (\mathbf{R} + \gamma \mathbf{P}_{a^*} \mathbf{V}^\pi) - (\mathbf{R} + \gamma \mathbf{P}_a \mathbf{V}^\pi) \tag{15}$$

$$= \gamma(\mathbf{P}_{a^*} - \mathbf{P}_a)\mathbf{V}^\pi \tag{16}$$

$$= \gamma(\mathbf{P}_{a^*} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_a)^{-1}\mathbf{R} \tag{17}$$

Finally, we write the original objective function (Eq. 9) in matrix form:

$$\arg\max_{\mathbf{R}} (\sum_s \min_a \{\gamma(\mathbf{P}_{a^*} - \mathbf{P}_a)(\mathbf{I} - \gamma \mathbf{P}_a)^{-1}\mathbf{R}\}) - \lambda||\mathbf{R}||_1 \tag{18}$$

Additional constraints: We can add another constraint (as in the original paper) by making use of the following theorem.

Theorem 1. *The future payoff of the best action is greater than that of another action.*

$$\sum_{s'} P(s'|s,a^*)V^{\pi}(s') \geq \sum_{s'} P(s'|s,a)V^{\pi}(s') \; \forall s,a \tag{19}$$

Proof.

$$Q^{\pi^*}(a^*,s) \geq Q^{\pi^*}(a,s) \tag{20}$$

$$r(s) + \gamma \sum_{s'} p(s'|s,a^*)V(s') \geq r(s) + \gamma \sum_{s'} p(s'|s,a)V(s') \tag{21}$$

$$p(s'|s,a^*)V(s') \geq \gamma \sum_{s'} p(s'|s,a)V(s') \tag{22}$$

Then, we convert the constraint (Eq. 19) to matrix form:

$$\mathbf{p}_{sa^*}\mathbf{V}_{\pi} \geq \mathbf{p}_{sa}\mathbf{V}_{\pi} \; \forall s,a \tag{23}$$

$$\mathbf{P}_{a^*}\mathbf{V}_{\pi} \succeq \mathbf{P}_{a}\mathbf{V}_{\pi} \; \forall a \tag{24}$$

$$\mathbf{P}_{a^*}(\mathbf{I}-\gamma\mathbf{P}_a)^{-1}\mathbf{R} \succeq \mathbf{P}_{a}(\mathbf{I}-\gamma\mathbf{P}_a)^{-1}\mathbf{R} \; \forall a \tag{25}$$

where $\succeq$ denotes element-wise inequality.

Linear Program Formulation: Based on Eq. 18 and 25, we have the linear program as follows:

$$\hat{\mathbf{R}} = \arg\max_{\mathbf{R}} (\sum_{s} \min_{a} \{\gamma(\mathbf{P}_{a^*} - \mathbf{P}_a)(\mathbf{I}-\gamma\mathbf{P}_a)^{-1}\mathbf{R}\}) - \lambda||\mathbf{R}||_1 \tag{26}$$

$$\text{s.t.} \mathbf{P}_a(\mathbf{I}-\gamma\mathbf{P}_a)^{-1}\mathbf{R} \; \forall a \neq a^* \tag{27}$$

$$|R(s)| \leq R_{max} \; \forall s \in S \tag{28}$$

which can be solved with a linear program routine.

2.3.4 IRL for large state spaces

Goal: Find the reward function(s) such that the policy is optimal.

Assumptions:

- State space can be very large or infinite.
- Transition model is known.
- Complete policy is known.

IRL for large state spaces assumes that the state space could be very large or infinite. This means that the reward function cannot be stored as a table. Therefore, we use an approximation function to model the reward function. In the lecture, we use a linear function as the approximation function of the reward:

$$R_{\theta}(s) = \theta_1\phi_1(s) + \theta_2\phi_2(s) + ... + \theta_d\phi_d(s) = \theta^T\phi(s) \tag{29}$$

Of course, we could also use a non-linear function, such as a deep network. However, such reward function cannot be solved by a LP, so we will keep using a linear function in the following discussion.

Now, if the reward function is a linear function of the state features, we can show that the value function can also be expressed as a linear function of expected state features.

Theorem 2. *Value function is a linear function of expected state features:*

$$V^\pi(s) = \theta^T \mu^\pi(s) \tag{30}$$

Proof. Begin from the definition of the value function,

$$V^\pi(s) = E_p[\sum_{t=0}^{\infty} \gamma r(s_t)] \tag{31}$$

$$= E_p[\sum_{t=0}^{\infty} \gamma \theta \cdot \phi(s_t)] \tag{32}$$

$$= \theta \cdot E_p[\sum_{t=0}^{\infty} \gamma \phi(s_t)] \tag{33}$$

$$= \theta \cdot \mu^\pi(s) \tag{34}$$

where $\mu^\pi(s) = E_p[\sum_{t=0}^{\infty} \gamma \phi(s_t)]$. □

Recall that the future payoff of the best action is greater than that of any other actions (as we proved in Theorem 1:

$$\sum_{s'} P(s'|s, a^*) V^\pi(s') \geq \sum_{s'} P(s'|s, a) V^\pi(s') \; \forall s, a \tag{35}$$

and we would like to make this difference as big as possible,

$$\max_\theta (E_{s' \sim p(s,a^*)}[V^\pi(s')] \geq E_{s' \sim p(s,a)}[V^\pi(s')]) \; \forall s, a \tag{36}$$

Note here the value function V depends on parameter θ. Because we have infinite number of states, we only optimize for a finite subset $s \in S_0 \subset S$. For actions, we only optimize for the second best action. This gives us the following objective function:

$$\max_\theta \min_a (E_{s' \sim p(s,a^*)}[V^\pi(s')] \geq E_{s' \sim p(s,a)}[V^\pi(s')]) \; \forall s \in S_0 \tag{37}$$

Finally, we can write the linear program like what we did in IRL for finite state spaces:

$$\max_\theta \sum_{s \in S_0} \min_a (E_{s' \sim p(s,a^*)}[V^\pi(s')] \geq E_{s' \sim p(s,a)}[V^\pi(s')]) \tag{38}$$

$$\text{s.t.} |\theta_d| \leq 1 \; \forall d \tag{39}$$

Note that in the original paper, there is an additional function g that helps the program to converge faster

$$\max_{\theta} \sum_{s \in S_0} \min_{a} g(E_{s' \sim p(s,a^*)}[V^{\pi}(s')] \geq E_{s' \sim p(s,a)}[V^{\pi}(s')]) \tag{40}$$

$$\text{s.t.} |\theta_d| \leq 1 \; \forall d \tag{41}$$

$$g(x) = \begin{cases} x, & x \geq 0 \\ 2x, & \text{otherwise} \end{cases} \tag{42}$$

The idea here is to put extra penalty when the future payoff of the best action is less than that of the other actions.

References

[1] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.